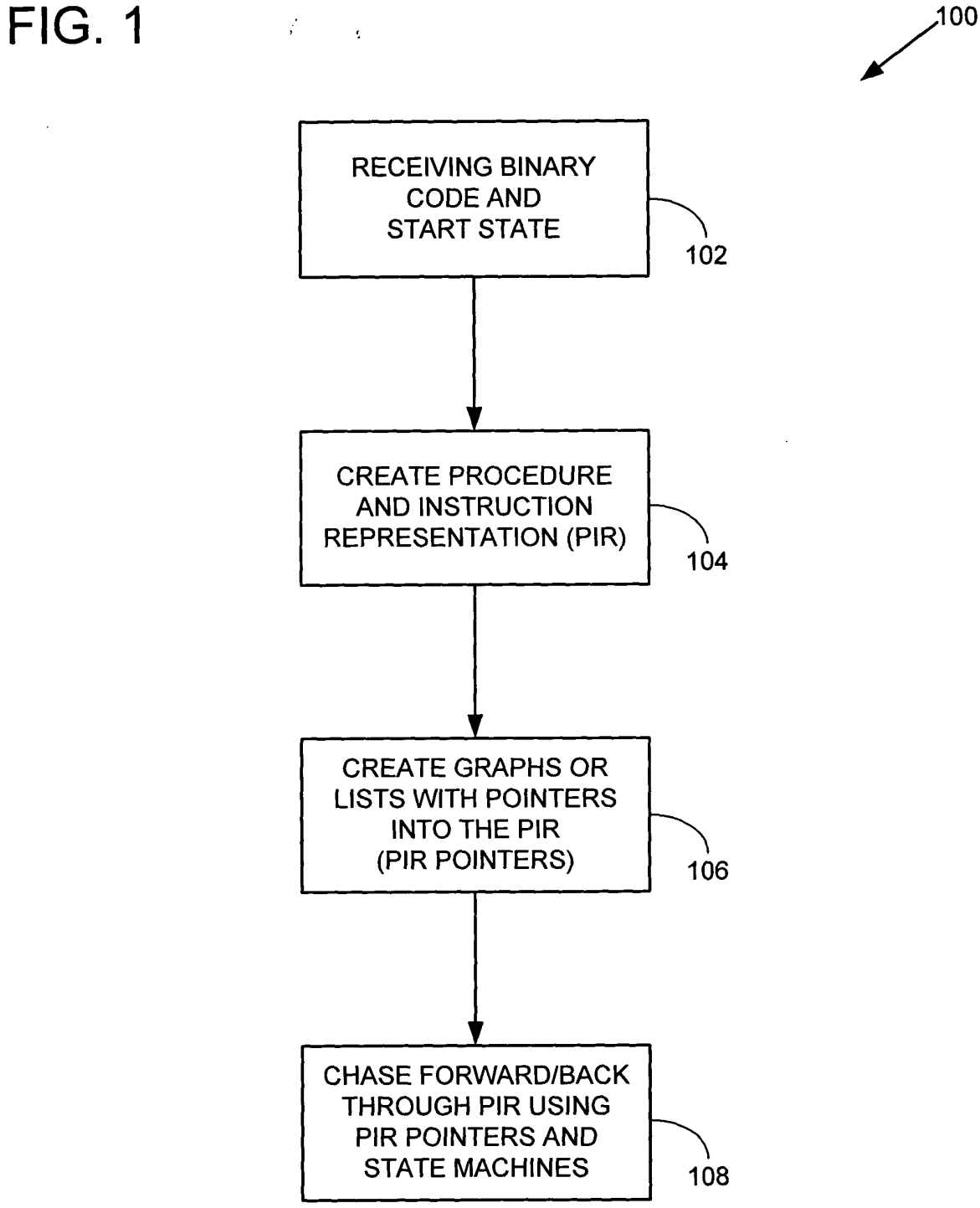
FIG. 1


100
RECEIVING BINARY CODE AND START STATE
102
CREATE PROCEDURE AND INSTRUCTION REPRESENTATION (PIR)
104
CREATE GRAPHS OR LISTS WITH POINTERS INTO THE PIR (PIR POINTERS)
106
CHASE FORWARD/BACK THROUGH PIR USING PIR POINTERS AND STATE MACHINES
108

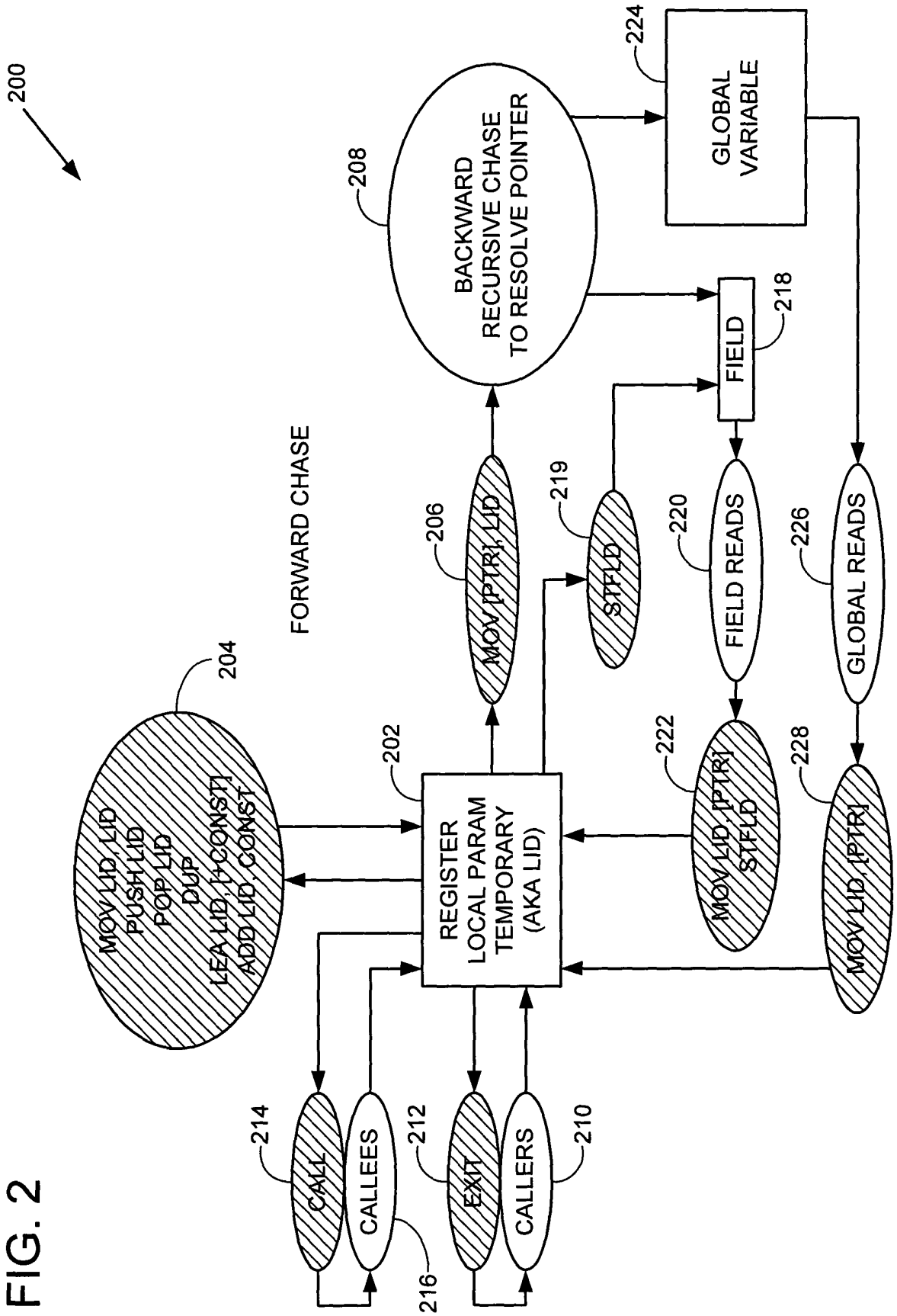

200
224
GLOBAL VARIABLE
208
BACKWARD RECURSIVE CHASE TO RESOLVE POINTER
218
FIELD
FORWARD CHASE
206
MOV [PTR], LID
219
STFLD
220
FIELD READS
226
GLOBAL READS
204
MOV LID, LID
PUSH LID
POP LID
DUP
LEA LID, [+CONST]
ADD LID, CONST
202
REGISTER LOCAL PARAM TEMPORARY (AKA LID)
222
MOV LID, [PTR]
STFLD
228
MOV LID, [PTR]
214
CALL
CALLEES
212
EXIT
CALLERS
210
216

FIG. 2

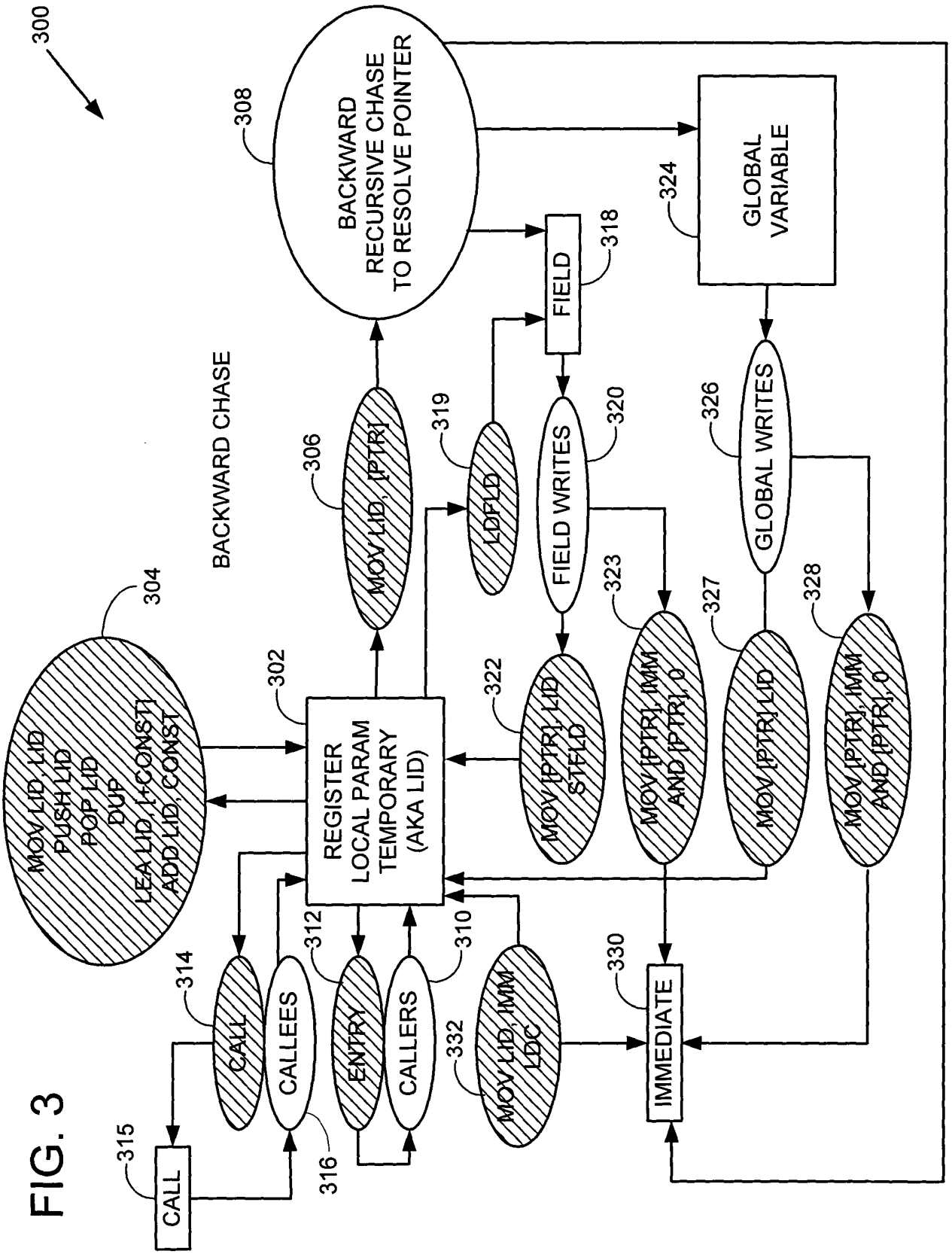

300
BACKWARD CHASE
308
BACKWARD RECURSIVE CHASE TO RESOLVE POINTER
GLOBAL VARIABLE
324
FIELD
318
306
MOV LID, [PTR]
319
LDFLD
320
FIELD WRITES
326
GLOBAL WRITES
304
MOV LID, LID
PUSH LID
POP LID
DUP
LEA LID, [+CONST]
ADD LID, CONST
302
REGISTER LOCAL PARAM TEMPORARY (AKA LID)
322
MOV [PTR], LID
STFLD
323
MOV [PTR], IMM
AND [PTR], 0
327
MOV [PTR], LID
328
MOV [PTR], IMM
AND [PTR], 0
314
CALL
312
ENTRY
310
CALLERS
330
IMMEDIATE
332
MOV LID, IMM
LDC
315
CALL
316
CALLEES

FIG. 3

FIG. 4

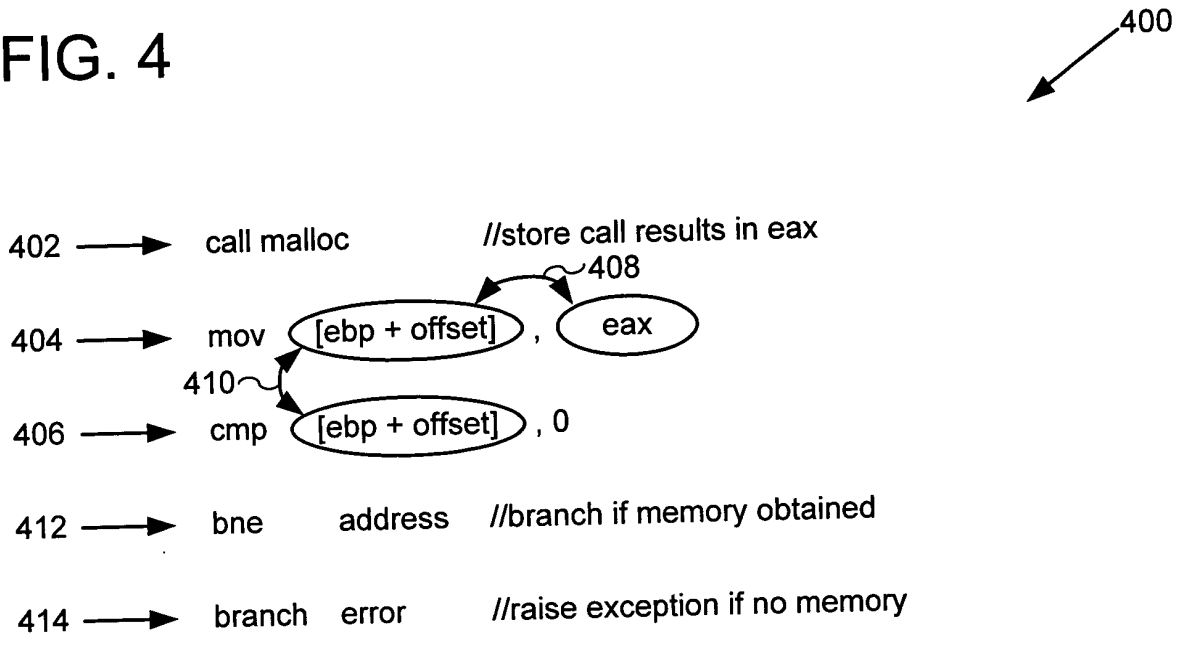

400
402
call malloc
//store call results in eax
408
404
mov
[ebp + offset]
,
eax
410
406
cmp
[ebp + offset]
, 0
412
bne
address
//branch if memory obtained
414
branch
error
//raise exception if no memory

FIG. 5

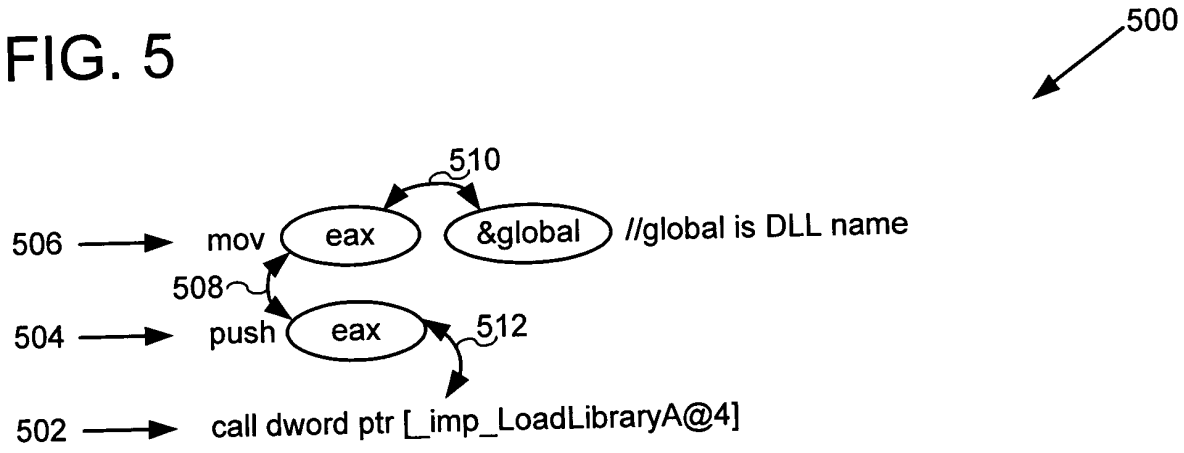

500
510
506
mov
eax
&global
//global is DLL name
508
504
push
eax
512
502
call dword ptr [_imp_LoadLibraryA@4]

FIG. 6

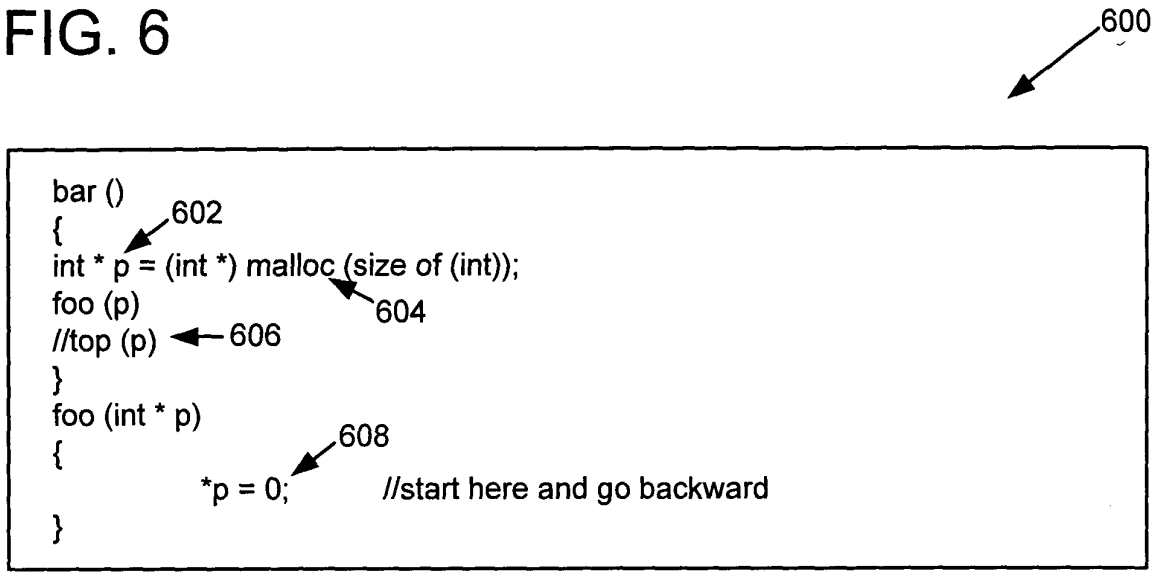
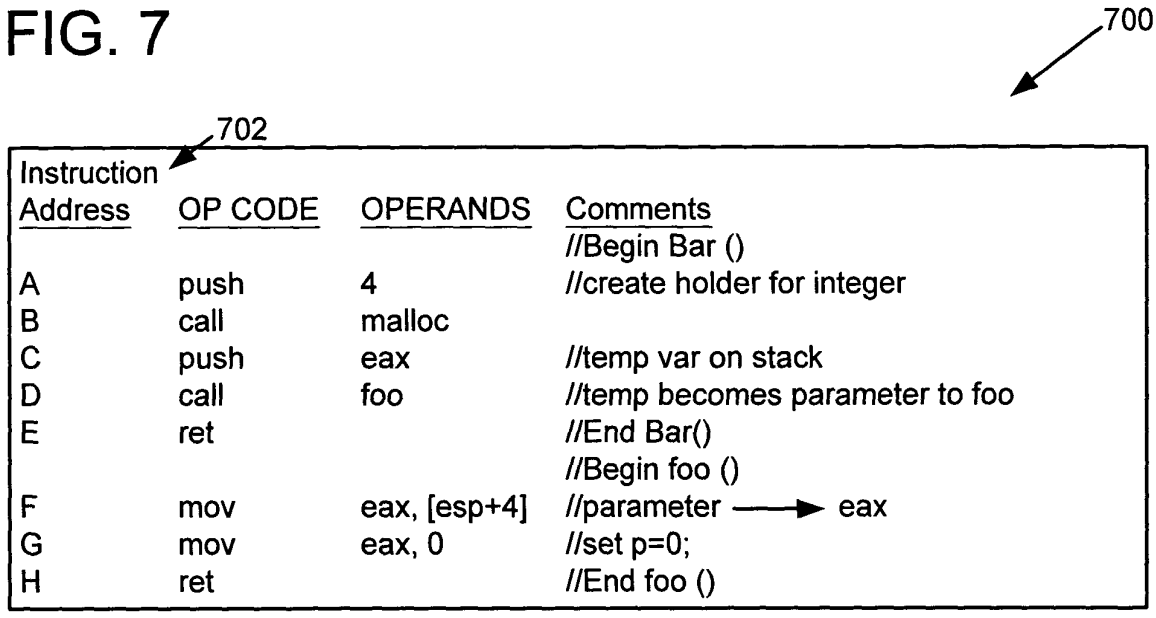


FIG. 7



Instruction Address	OP CODE	OPERANDS	Comments
			//Begin Bar ()
A	push	4	//create holder for integer
B	call	malloc	
C	push	eax	//temp var on stack
D	call	foo	//temp becomes parameter to foo
E	ret		//End Bar()
			//Begin foo ()
F	mov	eax, [esp+4]	//parameter ⟶ eax
G	mov	eax, 0	//set p=0;
H	ret		//End foo ()

FIG. 8

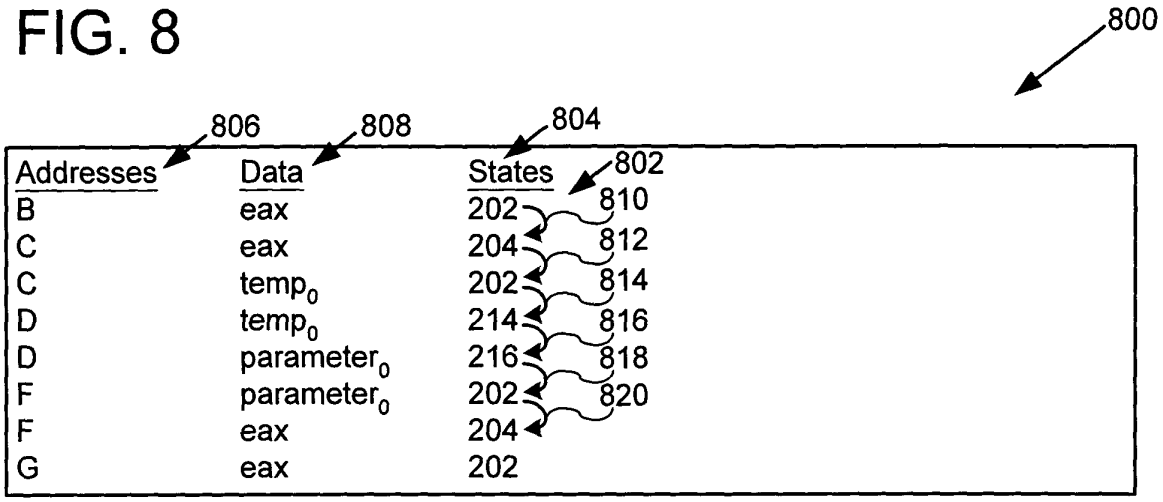

800
806
808
804
802
Addresses
Data
States
B
eax
202
810
C
eax
204
812
C
temp0
202
814
D
temp0
214
816
D
parameter0
216
818
F
parameter0
202
820
F
eax
204
G
eax
202

FIG. 9

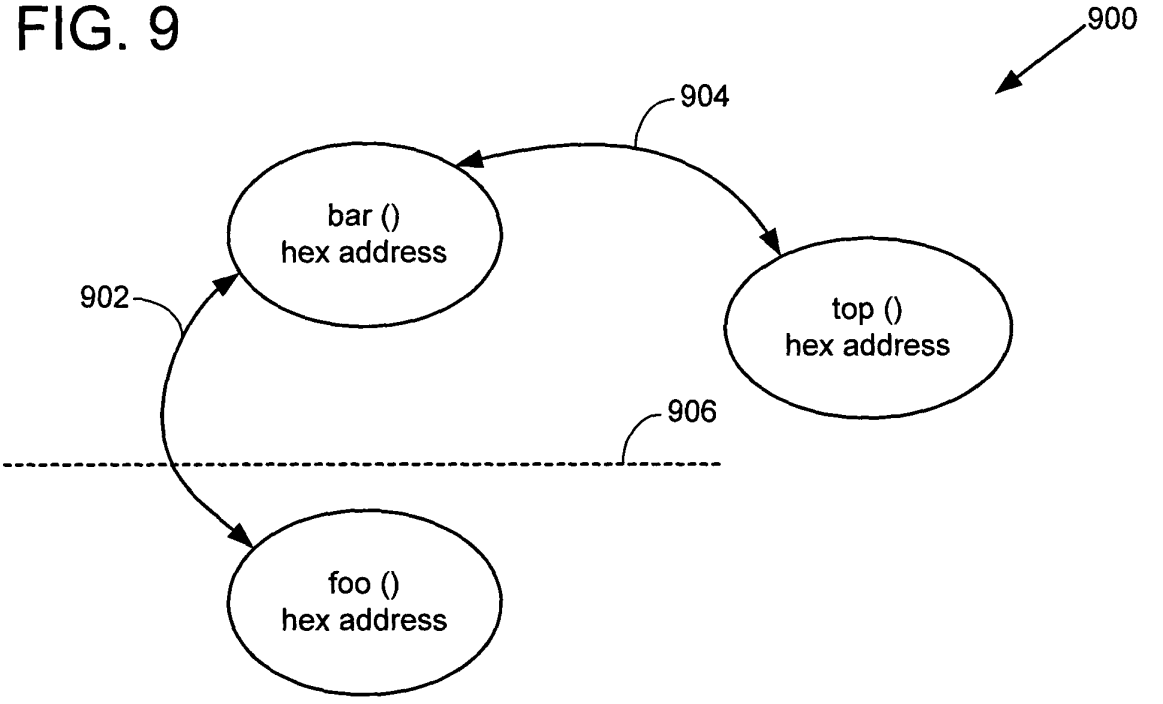
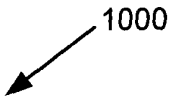
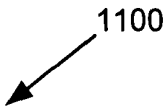

900
904
bar ()
hex address
902
top ()
hex address
906
foo ()
hex address

FIG. 10



Address	Data	State
G	eax	302
F	eax	304
F	$parameter_0$	302
D	$parameter_0$	312
D	$temp_0$	310
C	$temp_0$	302
C	eax	304
B	eax	302

FIG. 11



Addressing form (1102)	Resolves to (1103)
[global + imm] (1104)	Static data inside global (immediate or another global) If (global is written dynamically) (1106) Instructions that store into the global.

[global + scale * reg + imm] (1108)	if (reg can be chased to immediate)
	Static data inside global at offset (immediate or another global) (1110)
	If (global is written dynamically)
	Instructions that store into the global at offset.
	else
	All static data inside global (immediate or another global) (1114)
	If (global is written dynamically)
	Instructions that store into the global.

[reg + imm] (1116) (1118) (1120)	If (reg can be chased to global)
	data inside global at offset (immediate or global)
	If (global is written dynamically at offset)
	Instructions that store into the global at offset
	If (reg can be chased to type)
	instructions that store to field

[reg + scale * reg' + imm] (1122) (1124) (1128) (1126)	If (reg can be chased to global)
	if (reg' can be chased to immediate)
	data inside global at offset (immediate or global)
	If (global is written dynamically at offset)
	Instructions that store into the global at offset
	Else
	All data inside global (immediate or global)
	if (global is written dynamically)
	All instructions that store into the global
	If (reg can be chased to type)
	instructions that store to field array

FIG. 12

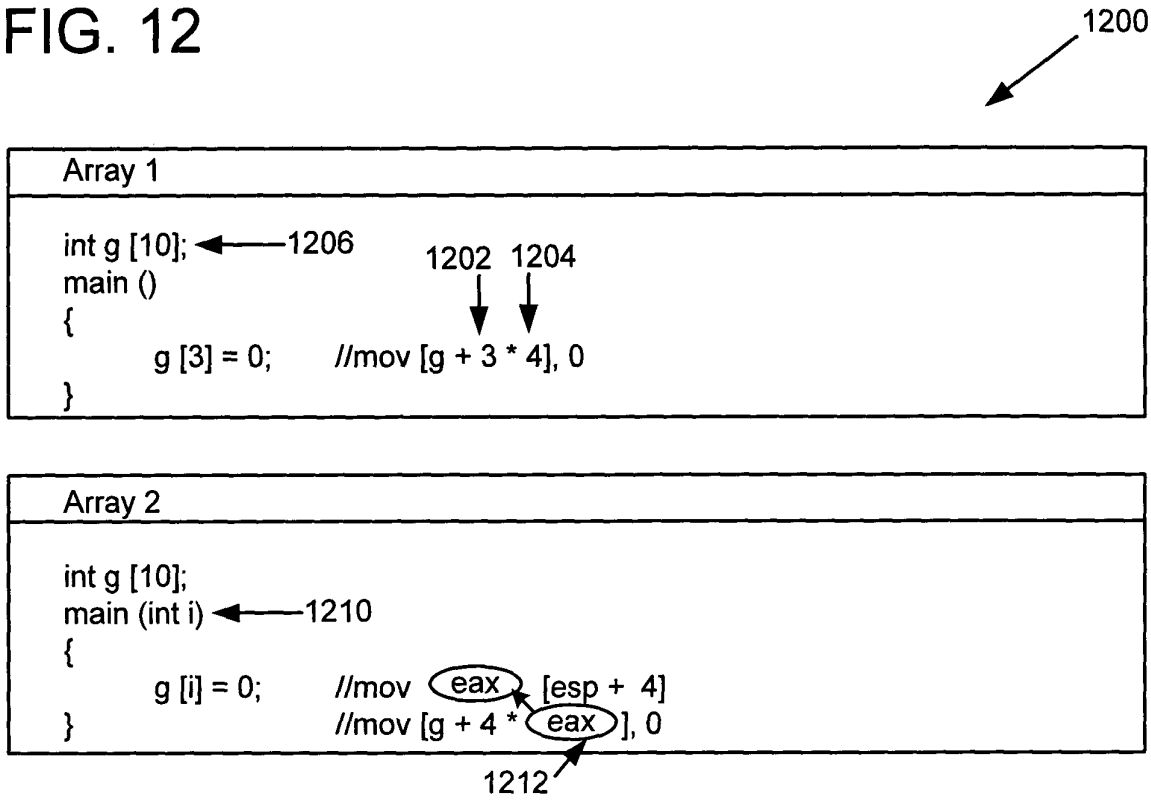

1200
Array 1
int g [10];
1206
main ()
{
g [3] = 0;
1202
1204
//mov [g + 3 * 4], 0
}
Array 2
int g [10];
main (int i)
1210
{
g [i] = 0;
//mov eax [esp + 4]
//mov [g + 4 * eax], 0
}
1212

FIG. 13

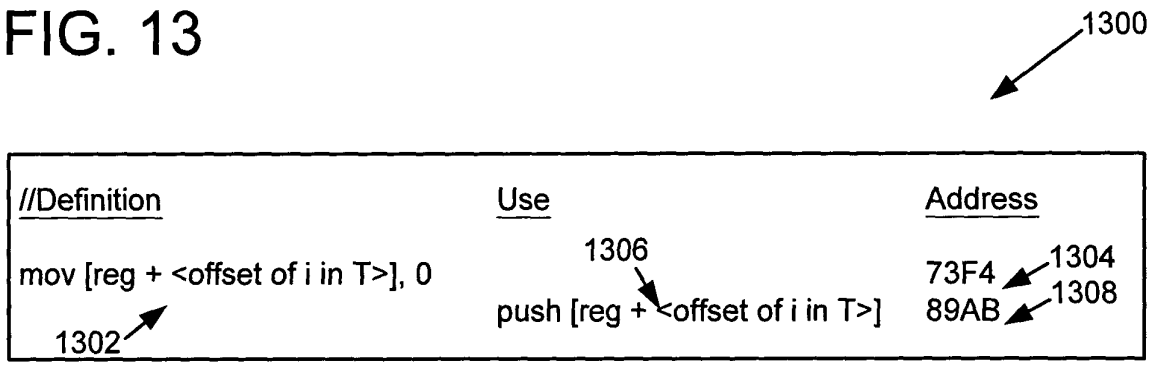

1300
//Definition
Use
Address
mov [reg + <offset of i in T>], 0
1306
push [reg + <offset of i in T>]
73F4
89AB
1304
1308
1302

FIG. 14

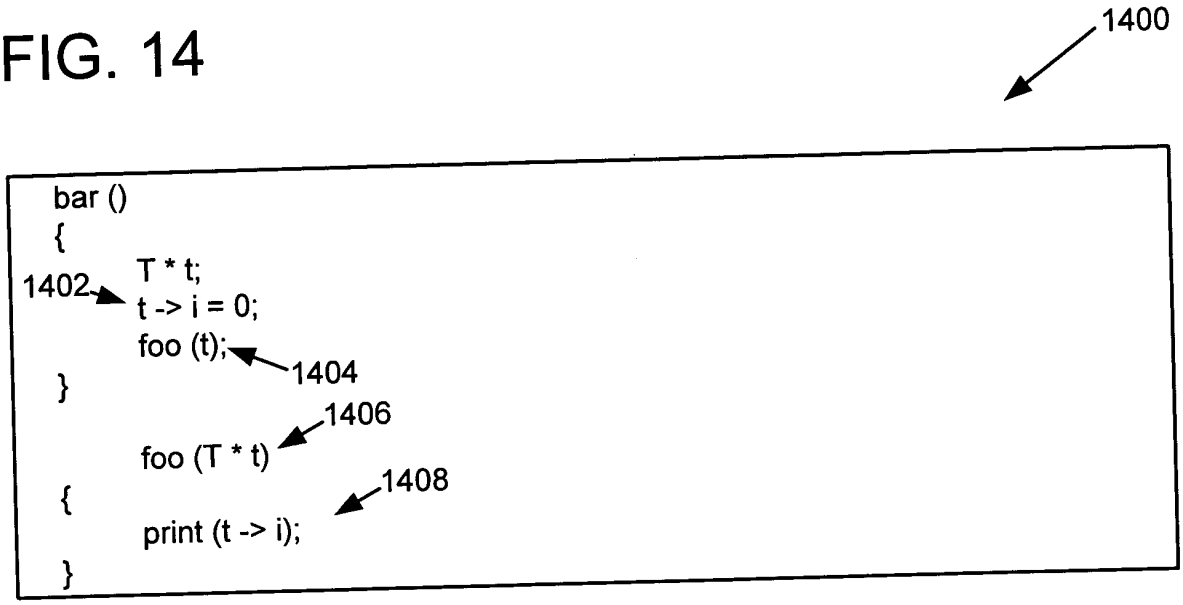

1400
bar ()
{
T * t;
1402
t -> i = 0;
foo (t);
1404
}
1406
foo (T * t)
1408
{
print (t -> i);
}

FIG. 15

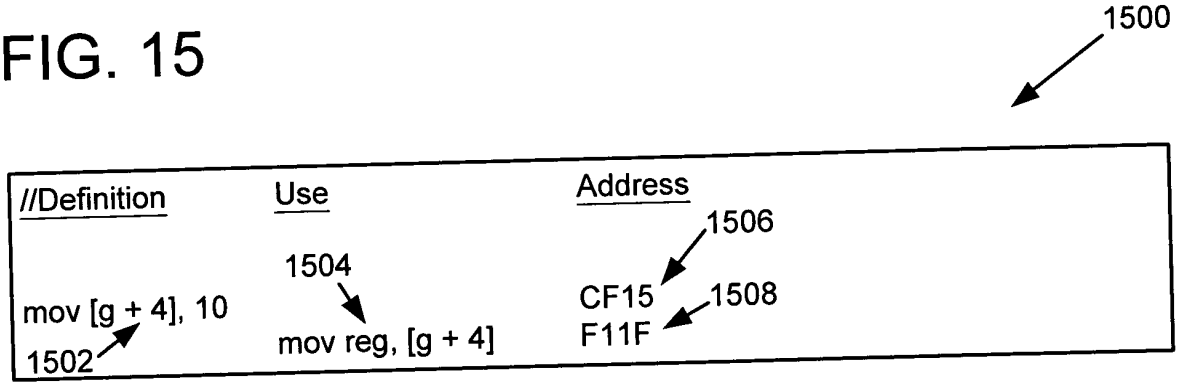

1500
//Definition
Use
Address
1506
1504
mov [g + 4], 10
mov reg, [g + 4]
CF15
1508
F11F
1502

FIG. 16

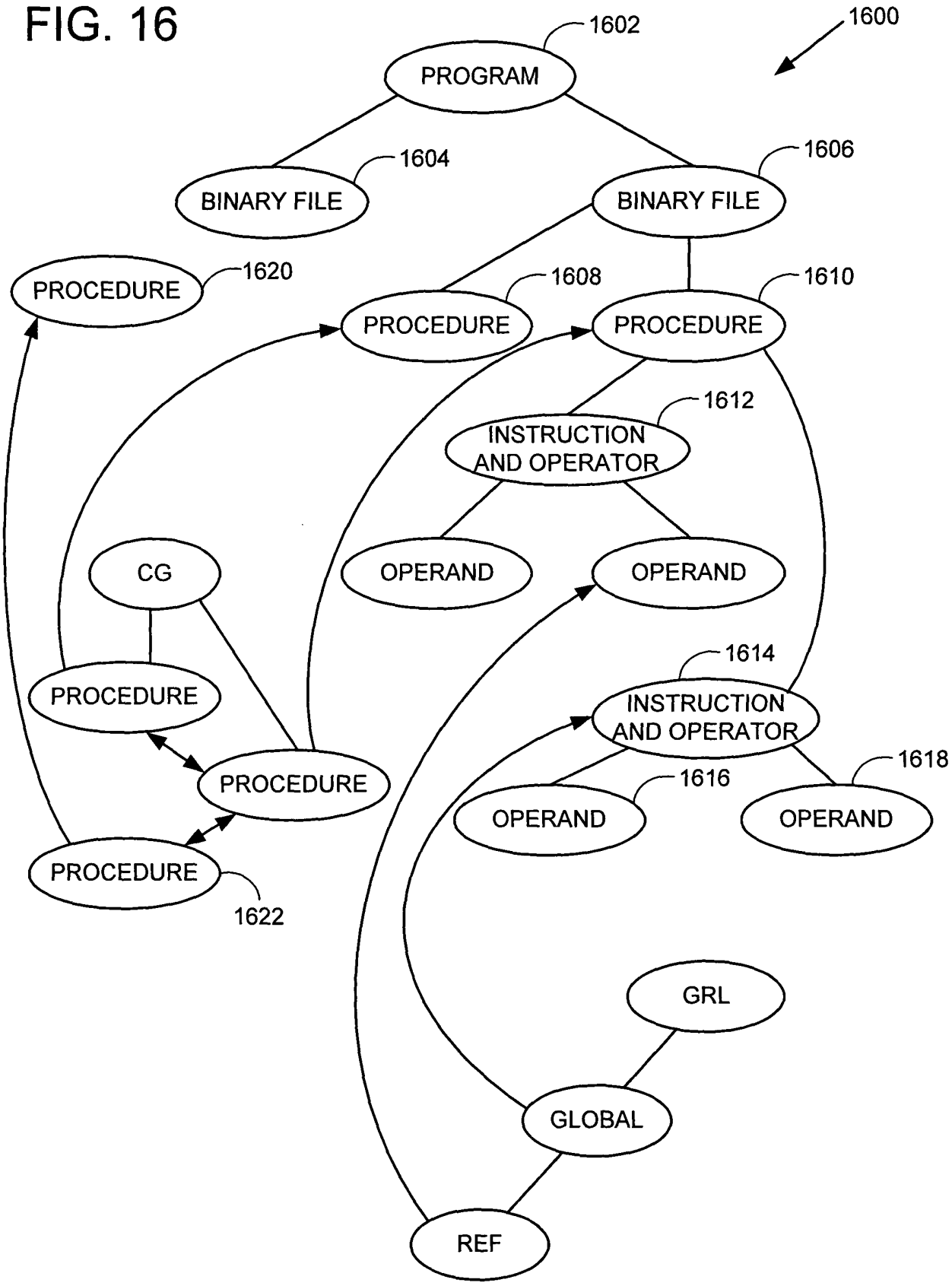

1600
1602
PROGRAM
1604
BINARY FILE
1606
BINARY FILE
1620
PROCEDURE
1608
PROCEDURE
1610
PROCEDURE
1612
INSTRUCTION AND OPERATOR
OPERAND
OPERAND
CG
PROCEDURE
1614
INSTRUCTION AND OPERATOR
1618
1616
OPERAND
OPERAND
PROCEDURE
PROCEDURE
1622
GRL
GLOBAL
REF

FIG. 17

```
/* Description:
 *   VChase is a class that can be used to follow data flow around a program
 */
class VChase                                          <-- 1702
{
public:
  // Create a chase object for some data at an instruction          <-- 1704
  static VULCANDLL VChase * VULCANCALL Create( VOperand op, VInst *pInst,
VProc *pProc, VComp *pComp );

  // Free memory assoicated with this object (and optionally the whole set)
  virtual void Destroy(bool fSet = true) = 0;          <-- 1706

  enum ChaseType                                       <-- 1708
  {
    ctRegister = 0,
    ctSymbol   = 1,
    ctGlobal   = 2,
    ctImmediate = 3,
    ctPointer = 4,
    ctArray = 5,
    ctDataMask = 7,
    ctLEA = 8,
    ctLEASymbol = ctLEA | ctSymbol,
    ctLEAGlobal = ctLEA | ctGlobal,
    ctLEAPointer = ctLEA | ctPointer,
    ctLEAArray = ctLEA | ctArray,
    ctReturn = 16,
    ctCantContinue = 32
  };

  // Get the current type of this chase object          <-- 1710
  virtual ChaseType Type() = 0;

  // Get the location of this chase object              <-- 1712
  virtual VInst *Inst() = 0;
  virtual VProc *Proc() = 0;
  virtual VComp *Comp() = 0;

  // Get the contents of this chase object              <-- 1714
  virtual ERegister Register() = 0;
  virtual VInstance *Instance() = 0;
  virtual VBlock *Global() = 0;
  virtual DWORD Immediate() = 0;
  virtual const VAddress *Pointer() = 0;
  virtual const VAddress *Array() = 0;
```

FIG. 18

```
// Does this object represent the return value from a call?
virtual bool IsCall() = 0;                                   <-- 1802

// Get the next chase object in this set
virtual VChase *Next() = 0;                                  <-- 1804

                                                             1806
// Chase across 1 thing and return set of new objects
virtual VChase *ChaseBackward() = 0;
virtual VChase *ChaseForward() = 0;

// Chase back to a symbol                                    1808
virtual VType *ChaseToType() = 0;
virtual VInstance *ChaseToInstance() = 0;

                                                             1810
// Chase until callback returns true
typedef bool (VULCANCALL *PFNCHASEDONE)(VChase *pCur);
virtual VChase *ChaseBackTo(PFNCHASEDONE) = 0;
virtual VChase *ChaseForwardTo(PFNCHASEDONE) = 0;

// Return type from IDone::Done (unavailable from static callback)
enum ChaseDone                                               1812
{
  cdContinueDiscard,
  cdDoneKeep,
  cdContinueKeepAsFrom,
  cdDoneDiscard,
};

// Chase using interface for callback
class IDone                                                  <-- 1814
{
public:
  virtual ChaseDone VULCANCALL Done(VChase *pCur) = 0;       <-- 1816
};
virtual VChase *ChaseBackTo(IDone * = NULL) = 0;             <-- 1818
virtual VChase *ChaseForwardTo(IDone * = NULL) = 0;

// Get the next node kept onlong the path
virtual VChase *From() = 0;

                                                             1820
// Predefined stopping routines for ChaseBackTo
static VULCANDLL bool VULCANCALL DoneAtType(VChase *);
static VULCANDLL bool VULCANCALL DoneAtImm(VChase *);
static VULCANDLL bool VULCANCALL DoneAtPointer(VChase *);
static VULCANDLL bool VULCANCALL DoneAtGlobal(VChase *);
static VULCANDLL bool VULCANCALL DoneAtLEA(VChase *);
static VULCANDLL bool VULCANCALL DoneAtCALL(VChase *);
};
```

FIG. 19

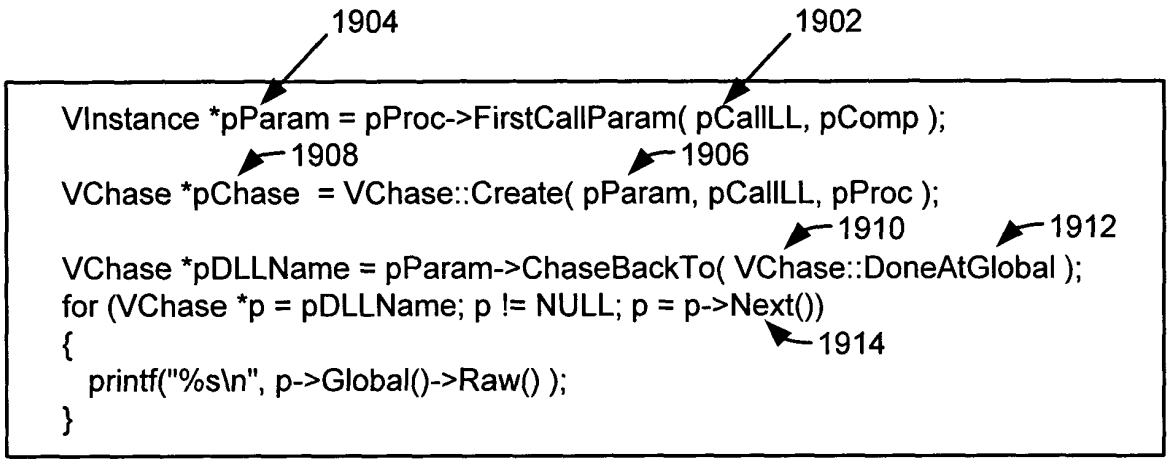

1904
1902
VInstance *pParam = pProc->FirstCallParam(pCallLL, pComp);
1908
1906
VChase *pChase = VChase::Create(pParam, pCallLL, pProc);
1910
1912
VChase *pDLLName = pParam->ChaseBackTo(VChase::DoneAtGlobal);
for (VChase *p = pDLLName; p != NULL; p = p->Next())
1914
{
printf("%s\n", p->Global()->Raw());
}

FIG. 20

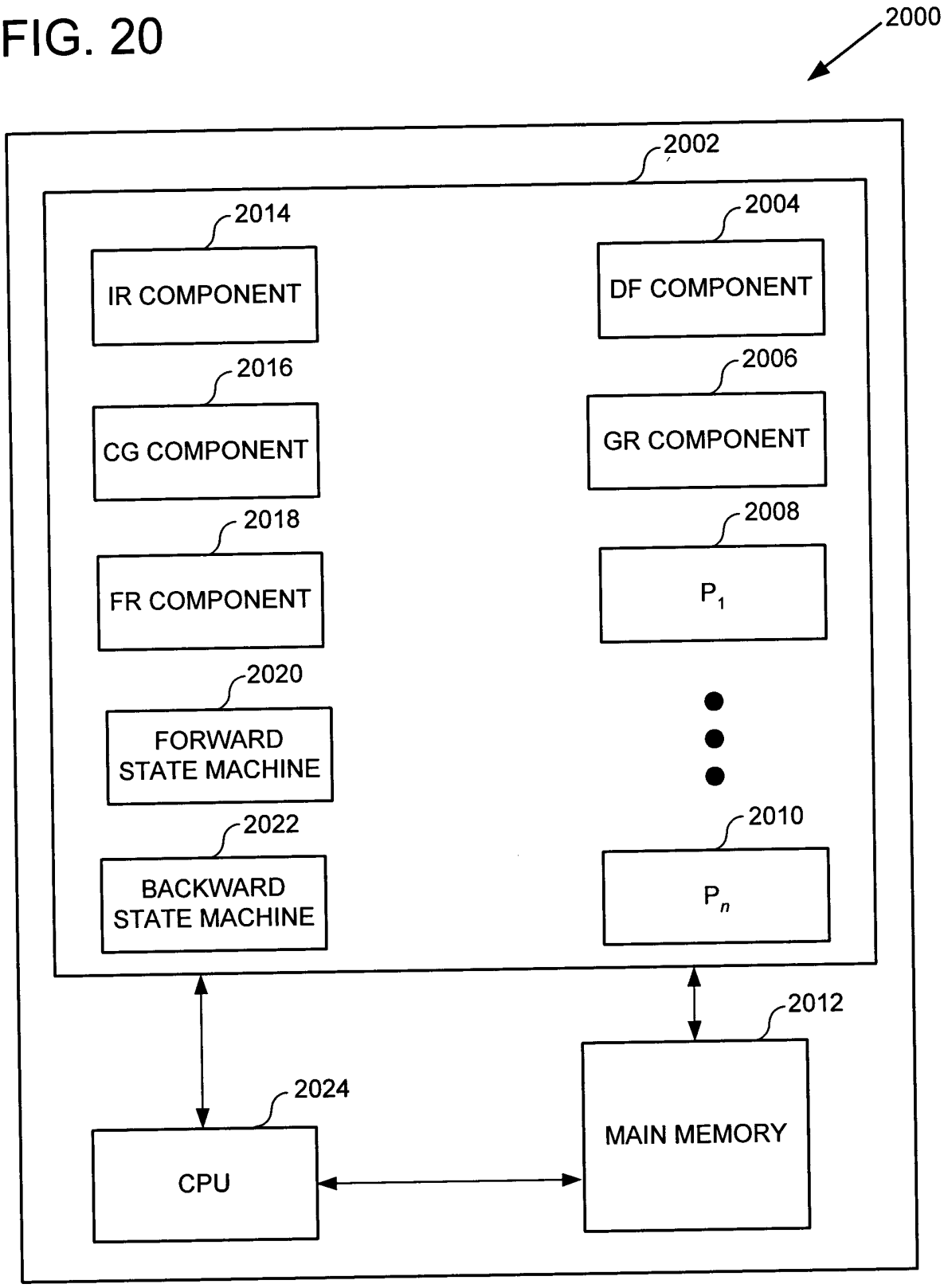
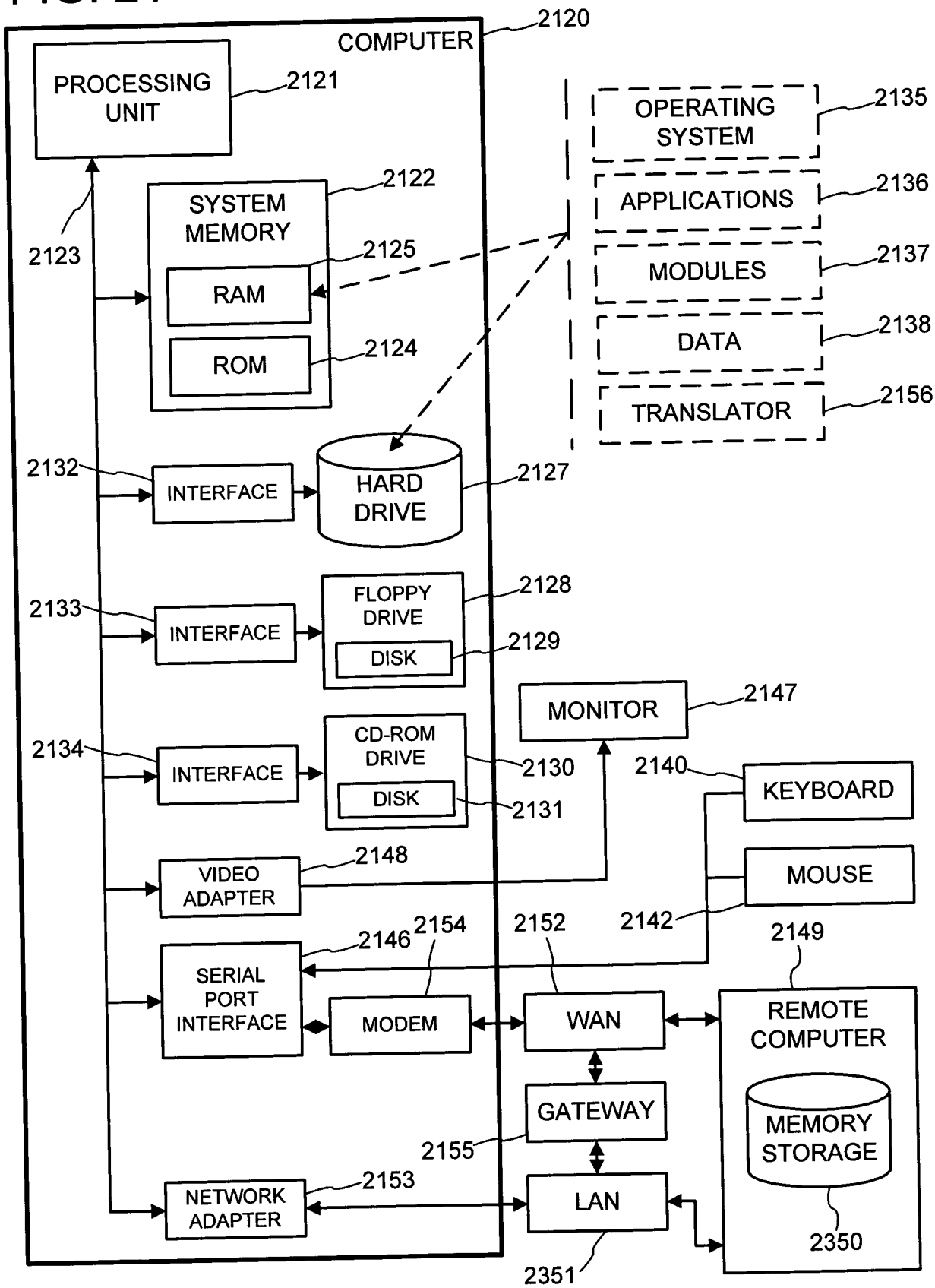

2000
2002
2014
IR COMPONENT
2004
DF COMPONENT
2016
CG COMPONENT
2006
GR COMPONENT
2018
FR COMPONENT
2008
P_1
2020
FORWARD STATE MACHINE
2022
BACKWARD STATE MACHINE
2010
P_n
2012
MAIN MEMORY
2024
CPU

FIG. 21


COMPUTER
2120
PROCESSING UNIT
2121
SYSTEM MEMORY
2122
2123
RAM
2125
ROM
2124
OPERATING SYSTEM
2135
APPLICATIONS
2136
MODULES
2137
DATA
2138
TRANSLATOR
2156
2132
INTERFACE
HARD DRIVE
2127
2133
INTERFACE
FLOPPY DRIVE
2128
DISK
2129
2134
INTERFACE
CD-ROM DRIVE
2130
DISK
2131
MONITOR
2147
2140
KEYBOARD
MOUSE
2142
2148
VIDEO ADAPTER
2146
2154
2152
2149
SERIAL PORT INTERFACE
MODEM
WAN
REMOTE COMPUTER
GATEWAY
MEMORY STORAGE
2155
2153
NETWORK ADAPTER
LAN
2350
2351